Buscador de Empleos Automatizado

Integrantes:
Cuch, Lucía Carolina
Kaplan, Azul
Stancato, Micaela

Docente:
Onna, Diego Ariel

Materia:
Taller de Programación II

Fecha de entrega:
27 de noviembre del 2025

Resumen

El presente proyecto consistió en el desarrollo de un **Buscador de Empleos Automatizado**, implementado como un Producto Viable Mínimo (MVP) de un *chatbot* para la plataforma Telegram. El objetivo principal fue resolver la problemática del proceso largo y tedioso de encontrar ofertas laborales que se ajusten al perfil de cada persona, y lograr evitar la pérdida de tiempo, frustración y dificultad para mantenerse al tanto de nuevas oportunidades laborales.

El bot interactúa con el usuario para recolectar cinco criterios de búsqueda (rubro, tipo de trabajo, nivel de experiencia, modalidad y ubicación). Posteriormente, utiliza la **ScrapingDog Jobs Search API** para extraer ofertas de LinkedIn en tiempo real. Finalmente, procesa y presenta los resultados, añadiendo un valor agregado crucial: un análisis y priorización de las mejores ofertas realizado por **Gemini AI**.

Objetivos

El desarrollo del Buscador de Empleos Automatizado se centró en los siguientes objetivos específicos:

- **Automatizar la búsqueda de empleo** para el usuario, simplificando el proceso largo y tedioso de encontrar ofertas laborales acordes a su perfil.

- **Implementar un flujo conversacional guiado** en Telegram que recoja de forma secuencial y validada los cinco parámetros clave de búsqueda: rubro, tipo de trabajo, nivel de experiencia, modalidad y ubicación.

- **Integrar APIs de terceros** para la funcionalidad central del bot:
    - Utilizar la **ScrapingDog Jobs Search API** para extraer ofertas laborales de LinkedIn en tiempo real.
    - Integrar **Gemini AI** para analizar, resumir y priorizar las ofertas, añadiendo un valor agregado al listado crudo de resultados.

- Modularizar el código aplicando la **Programación Orientada a Objetos (POO)** mediante clases específicas (GestorSolicitudes, ExtractorEmpleos, ProcesadorResultados, Presentador) para una gestión de flujo de datos y lógica clara.

- **Garantizar la estabilidad del MVP** mediante la implementación de Manejo de Excepciones (try...except) para fallos críticos de red o API, registrando estos errores en un log_errores.txt.

Tecnologías utilizadas

La solución se implementó utilizando un conjunto de herramientas y metodologías de programación esenciales para el desarrollo de *chatbots*:

Tipo de Tecnología	Componente	Descripción y Uso Específico
Arquitectura de Software	**Programación Orientada a Objetos (POO)**	Estructuración del código en clases para encapsular responsabilidades y facilitar el manejo del estado de la conversación.
Herramientas de Programación	**Manejo de Excepciones**	Implementación de `try...except` para capturar fallos de red (`requests.exceptions.HTTPError`) y de la API de IA, registrando errores en `log_errores.txt`.
API de Extracción de Datos	**ScrapingDog Jobs Search API**	Utilizada para extraer ofertas laborales en tiempo real de LinkedIn, proporcionando los resultados segmentados.
API de Inteligencia Artificial	**Gemini AI**	Integrada para el análisis avanzado y la priorización de las ofertas, generando un informe de valor para el usuario.
Framework de Telegram	***python-telegram-bot***	Permite el polling y la interacción del bot. Configurado con parse_mode='HTML' para la presentación estable del formato.
Lenguaje de Programación	**Python**	Lenguaje principal de implementación, elegido por su versatilidad en la integración de APIs y su ecosistema de librerías para el desarrollo de bots.

Descripción de la solución / MVP

El Buscador de Empleos Automatizado es un *chatbot* de Telegram diseñado para actuar como un asistente inteligente que filtra y analiza oportunidades laborales activas.

Flujo operacional

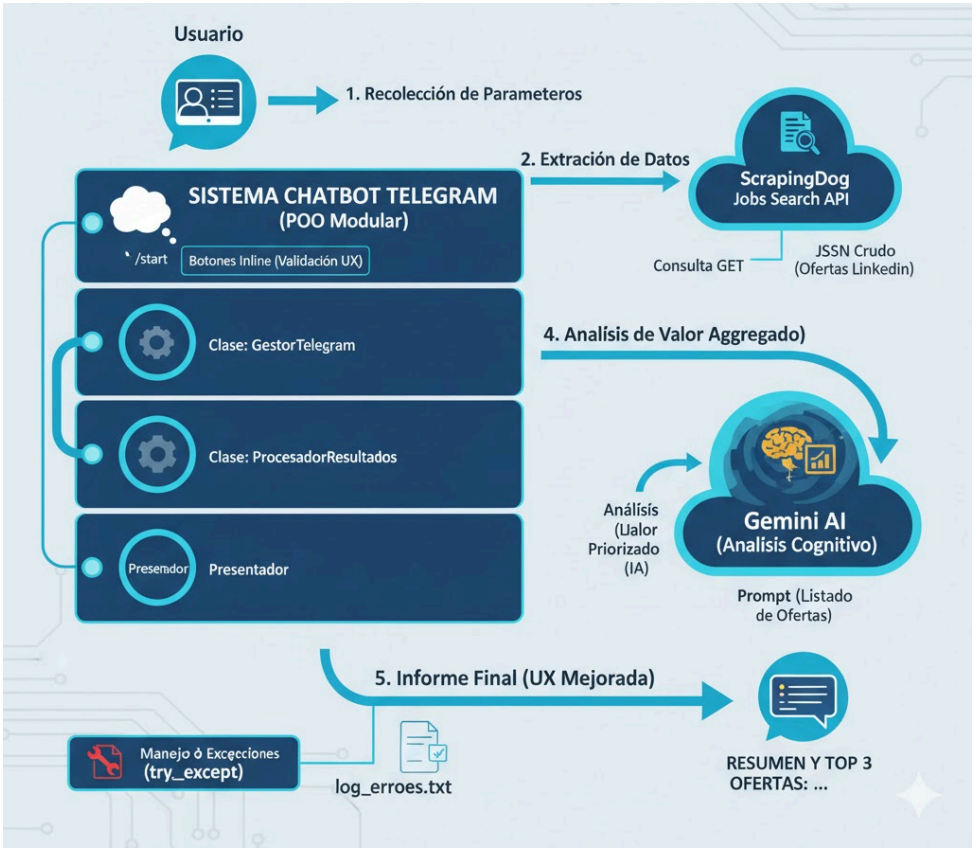
1. **Inicio y Recolección de criterios:** El usuario inicia la búsqueda con /start. El Gestor de Solicitudes guía al usuario paso a paso para recolectar los 5 parámetros requeridos:
    - **Texto libre:** Rubro o área de trabajo y Ubicación geográfica.
    - **Botones Inline:** Tipo de trabajo, Nivel de experiencia y Modalidad (ej: Remoto, Híbrido), garantizando datos válidos para la API.

2. **Extracción de datos:** Con los parámetros completos, el ExtractorEmpleos realiza una solicitud GET a la ScrapingDog API. La API devuelve los resultados crudos de las ofertas laborales de LinkedIn que cumplen con todos los criterios.

3. **Procesamiento y Presentación cruda:** El ProcesadorResultados normaliza los datos. Luego, el Presentador muestra inmediatamente un Listado Crudo de las primeras 5 ofertas encontradas, incluyendo Título, Empresa, Ubicación y el enlace a la publicación original en un formato HTML.

4. **Análisis de Valor Agregado (Gemini AI):** Inmediatamente después del listado, el Presentador interactúa con Gemini AI. La IA analiza un conjunto de las ofertas y genera un informe final que resume el mercado y prioriza las 3 a 5 mejores opciones con una breve justificación.

Diagrama:



Estructura del código

Aunque el diseño inicial se concibió bajo una arquitectura modular de múltiples archivos (un archivo por cada clase principal: GestorTelegram, ExtractorEmpleos, ProcesadorResultados y Presentador), la solución final unificó todas estas clases en el único archivo. Esta decisión de refactor simplificó la gestión del estado conversacional (user_data) y la lógica de debugging, manteniendo aun así la encapsulación y la separación de responsabilidades a través de clases de Programación Orientada a Objetos (POO).

1. **Clase GestorTelegram (El controlador de la conversación)**
    - **Rol:** Actúa como el controlador principal del chatbot. Es responsable de iniciar el flujo, enviar las peticiones de datos al usuario y manejar la lógica de estado de la conversación.
    - **Manejo de estado:** Utiliza un diccionario global (user_data) para encapsular y rastrear los parámetros de búsqueda de cada usuario (chat_id), asegurando que la información de una sesión no interfiera con otra.
    - **Métodos clave:** pedir_rubro(), pedir_tipo(), pedir_nivel(), etc., que se encadenan secuencialmente.

2. **Clase ExtractorEmpleos (La capa de servicios externos)**
    - **Rol:** Encapsula toda la lógica relacionada con la conexión y la consulta a la ScrapingDog Jobs Search API.

- **Responsabilidad:** Construye dinámicamente la URL de la API con los cinco parámetros recolectados y ejecuta la solicitud GET utilizando la librería requests.
- **Manejo de Excepciones:** Incluye bloques try...except para manejar posibles requests.exceptions.HTTPError o fallos de conexión, devolviendo un resultado de error controlado.

3. **Clase ProcesadorResultados (La capa de transformación de datos)**
   - **Rol:** Su única responsabilidad es recibir el JSON crudo devuelto por el ExtractorEmpleos y normalizarlo.
   - **Procesamiento:** Itera sobre la lista de resultados y filtra la información relevante (título, empresa, ubicación, link) para crear una estructura de datos más limpia y utilizable, lista para ser presentada o analizada por la IA.

4. **Clase Presentador (La capa de output y valor agregado)**
   - **Rol:** Gestiona la salida de información al usuario. Es el punto de integración con Gemini AI.
   - **Métodos clave:**
     - mostrar_resultados_crudos (chat_id, resultados): Formatea la lista de ofertas recibidas desde ProcesadorResultados en formato HTML de Telegram (utilizando etiquetas <b>, <i>, y <a href> para los enlaces) para su correcta visualización y estabilidad.
     - mostrar_analisis_ia (chat_id, respuesta_ia): Envía el informe estratégico generado por Gemini AI, aplicando la lógica de fragmentación de mensajes si la respuesta supera el límite de caracteres de Telegram.

5. **Función log_error(mensaje)**
   - **Rol:** Utilidad fundamental de robustez. Es llamada por el método GestorSolicitudes.ejecutar_busqueda cuando ocurre una excepción crítica (ej. fallo de API o de red).
   - **Función:** Registra el mensaje de error junto con la fecha y hora en el archivo log_errores.txt, permitiendo el monitoreo del funcionamiento del bot sin depender de la consola.

Dificultades encontradas y soluciones

1. **Fragmentación del flujo:** Inicialmente, el código se desarrolló en múltiples archivos modulares. Al intentar unificar los *scripts* de Telegram y la IA, la dispersión del código y la gestión de variables de estado se volvieron ineficientes.

   Solución: Se tomó la decisión de unificar todo el código en un único archivo. Esta refactorización simplificó el *debugging* y mejoró la visibilidad, manteniendo la POO mediante la definición clara de las clases.

2. **Validación de Datos y UX:** Inicialmente se solicitaban todos los parámetros como texto libre, lo que generaba errores al enviar valores no válidos a la API externa.

Solución: Se implementó el uso de botones *inline* para los campos Tipo, Nivel y Modalidad. Esta mejora de la Experiencia de Usuario (UX) garantiza la validación automática de datos, ya que solo se envían los valores predefinidos que la ScrapingDog API puede procesar.

3. **Markdown Inconsistente de IA y mensajes largos:** El bot inicialmente usaba el formato Markdown. El texto generado por Gemini AI a veces incluía formatos que entraban en conflicto con el analizador de Telegram, resultando en errores de tipo *Bad Request: can't parse message text*. Adicionalmente, mensajes muy largos superaban el límite de 4096 caracteres.

   Solución (Formato): Se configuró el framework de Telegram para utilizar parse_mode='HTML' de forma global en la inicialización del bot. Esto proporcionó una mayor estabilidad y consistencia en el formato de todos los mensajes.

   Solución (Mensajes largos): Se implementó una lógica de fragmentación de mensajes en la clase Presentador que divide la respuesta de la IA en bloques de menos de 3500 caracteres, asegurando la entrega completa del informe al usuario sin caer en el error de límite de Telegram.

Conclusiones y perspectivas

El proyecto del Buscador de Empleos Automatizado ha demostrado ser un Producto Viable Mínimo (MVP) exitoso, logrando su objetivo central de automatizar y agregar valor al proceso de búsqueda laboral. El éxito del proyecto reside en la orquestación eficiente de tecnologías de terceros (Telegram, ScrapingDog API y Gemini AI) bajo una estructura de código robusta.

Se cumplió el objetivo académico de aplicar la Programación Orientada a Objetos (POO) de forma sólida, aunque la estructura final se unificó en un único archivo, manteniendo la POO a través de clases bien definidas. Esta encapsulación permitió gestionar el estado conversacional de forma segura y simplificar el mantenimiento futuro. Además, se logró la integración fluida de dos APIs externas con propósitos distintos: una para la extracción de datos en tiempo real (ScrapingDog) y otra para el análisis cognitivo y valor agregado (Gemini AI).

La implementación rigurosa del Manejo de Excepciones (try...except) y la función de logging (log_error) garantizan la estabilidad del servicio; el bot puede tolerar fallos en las APIs externas o problemas de red sin caer.

Es importante destacar que la decisión de refactorizar el flujo conversacional migrando de texto libre a botones inline resolvió un problema crítico de validación de datos, mejorando drásticamente la Experiencia de Usuario (UX) al hacer el bot intuitivo. El uso de Gemini AI transformó el producto de un simple motor de búsqueda a un asistente de selección de personal. Al generar un resumen y una priorización de las mejores ofertas, el bot no solo entrega datos, sino que proporciona una recomendación estratégica, cumpliendo con la consigna de generar un valor agregado significativo.